FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Technical Report

Performance of single-processor BLAS on IBM p690

Inge Gutheil

FZJ-ZAM-IB-2004-08

August 2004

(last change: 10.08.2004)

Performance of single-processor BLAS on IBM p690

Inge Gutheil

John von Neumann-Institut für Computing
Zentralinstitut für Angewandte Mathematik
Forschungszentrum Jülich GmbH
August 10, 2004

Abstract

The Basic Linear Algebra Subprograms, BLAS, are the basic computational kernels in most applications. BLAS 1 and BLAS 2, the vector-vector and matrix-vector routines, require memory accesses in the same order as computations and thus cannot achieve performance close to peak performance on modern computer architectures. BLAS 3 matrix-matrix operations on $n \times n$-matrices on the other side can do order n^3 operations with only order n^2 memory accesses. This much better ratio of computation to memory access allows for much higher performance. To show which performance can be expected using the BLAS routines from IBM's ESSL on an IBM p690 we investigated the performance of one routine of each BLAS level and compared it to that of the corresponding routines on a CRAY T3E.

1 Introduction

The Basic Linear Algebra Subprograms, BLAS ([1], [2], [3]), have been the basic building blocks for application programs since the 1980th with the upcoming vector machines and even more since the 1990th with the BLAS 3 being optimally suited for computers with slow memories and fast CPUs and caches. Almost all hardware vendors deliver optimized BLAS routines for their architectures and with the ATLAS ([5]) project there are self-adapting optimized BLAS for any computer with a C and a Fortran compiler.
An important factor for performance on modern computer architectures with relatively slow memory access compared to CPU speed and access to cache is the so called "data re-use factor" r (see [6]) which is the number of operations performed divided by the amount of data moved from the slowly accessible main memory to the fast cache running almost at full processor speed.
For BLAS 1 calculations r is approximately 1. For the so-called AXPY operation ($\vec{y} = \alpha\vec{x} + \vec{y}$) r is $2/3$. Here $2n$ data, namely the two vectors $\vec{y}$ and $\vec{x}$, have to be loaded and $\vec{y}$ has to be stored, which makes $3n$ data movements. On these data n multiplications and n additions, which makes $2n$ floating-point operations, are performed.
For BLAS 2 operations r is about 2, e.g. for GEMV, the matrix-vector-multiply routine ($\vec{y} = \alpha A\vec{x} + \beta\vec{y}$). Here the matrix A (order of n^2 data), has to be transferred from memory and about n^2 multiplications and n^2 additions ($2n^2$ floating-point operations) are performed on these data.
Only for BLAS 3 operations r is growing with the matrix size. r is approximately n/2 for GEMM, the matrix-matrix-multiplication ($C = \alpha AB + \beta C$) which performs $2n^3$ operations and has to do $3n^2$ loads for the three matrices and n^2 stores for matrix C.
It can be seen from the data re-use factor, that only BLAS 3 operations can fully exploit the processor speed if problems are large enough, whereas for BLAS 1 and BLAS 2 operations performance is limited by memory access speed.
We measured the performance of three BLAS routines, one of each level, from the ESSL library from IBM and compared it to peak performance and to the performance of the corresponding routines from libsci on one processor of CRAY T3E, which also has a main memory to which access is much slower than the CPU speed and a cache which delivers fast data access only if data can be re-used.
On IBM p690 we had to measure each problem size several times as the performance varied much (see figure 1). To see which performance is possible we took the best result of several runs for the performance diagrams. On CRAY T3E the results were more reproducible, so we most often did only one measurement per problem size. Where we did more than one measurement we also took the best value.

2 Characteristics of the machines used

The Juelich Multi Processor Jump [7] consists of 41 IBM p690 frames, the nodes, each node with 32 processors Power4+, 1.7 GHz. Each processor has two FPUs with a multiply-add instruction. This means that one processor can do up to four floating-point operations per cycle which results in a peak performance of $4 \times 1.7 = 6.8$ GFLOPS for each processor.
Each processor has 64 KByte instruction and 32 KByte data internal level 1 cache, every two processors share 1.5 MByte level 2 cache with 10-12 cycles latency and every 32 processors share 512 MByte level 3 cache with 92-100 cycles latency. Each frame of 32 processors has 128 GByte shared main memory running at 567 MHz and with a latency of 252 cycles. It can deliver 3.5 GByte/second.
The tests were executed under the operation system AIX 5.2 with ESSL V4.1. The Fortran compiler was XL FORTRAN version 8.1 with compiler option -O3.
The CRAY T3E also has a floating-point multiply-add unit and thus can reach a peak performance of 1200 MFLOPS with a processor speed of 600 MHZ. The processors on CRAY T3E do not share any memory which means that load on other processors does not influence the performance on one processor. Each processor has 512 MByte of main memory and the cache size is 96 KByte out of which 32 KByte can be used for the application.
The tests on CRAY T3E were executed under Unicos/mk 2.0.6.07. with programming environment PrgEnv.36. The compiler was f90 with option -O3.

3 Performance measurements

We measured execution times of the library routines DAXPY, DGEMV, and DGEMM from ESSL on Jump with the timing routine rtc() and SAXPY, SGEMV, and SGEMM from libsci on CRAY T3E with system_clock. To find the possible performance bottlenecks we measured the times for vector lengths and matrix sizes which are multiples of 50 as well as those which are multiples of 32, as the latter often cause performance degradation on computers with caches. The measurements were done in loops for $n = nmin$ to $n = nmax$ by steps of $ninc$ (see Appendix), where n is the vector length and matrix size.
The execution times of all routines are rather small for small problem sizes. Thus we measured the times in a loop repeating the call to the routines $maxtest$ times and computed the execution time for one execution by dividing by $maxtest$. $Maxtest$ was chosen to deliver an execution time for the loop in the order of 0.1 sec to 1 sec. For longer execution times $maxtest = 1$ was selected.
We additionally computed the MFLOPS by assuming that an AXPY-operation on a vector of length n requires $2n$ floating point operations, a GEMV-operation on a matrix of size $n \times n$ and a vector of length n needs $2n^2$, and a GEMM-operation on matrices of size $n \times n$ needs $2n^3$ floating point operations.

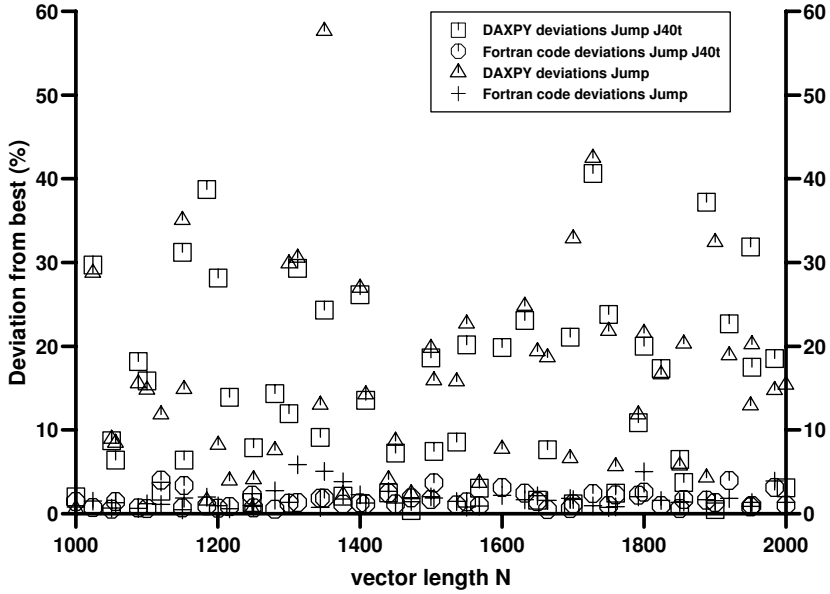

Figure 1: Performance of DAXPY on one processor of IBM p690. For matrix sizes between 1000 and 2000 the deviation of the lowest MFLOP rate from the highest MFLOP rate in percent is shown. Additionally the same values are shown for a Fortran coded AXPY loop. Jump J40t means a separate part of Jump with the same environment as the rest but without other users.

Figure 1 shows that, using the library routine DAXPY, on IBM p690 for small vector lengths there are sometimes large differences between the smallest and the highest MFLOP rate for the same problem size. To find out whether this is due to other users on the same node sharing the resources we did the measurements also on a dedicated part of Jump called J40t, but the same effects could be seen there.
The effect of rather large deviations of the slowest execution time from the best one could be seen for all BLAS routines and all problem sizes on IBM p690, but for higher order BLAS routines the deviations were overall smaller than for the AXPY routine. On CRAY T3E the deviations from the best performance were rather small, each processor is really separated from all other processors and users. Thus we decided to do the measurements on IBM p690 at least three times for each problem size whereas we often did only one measurement per problem size

on CRAY T3E.
For the case of AXPY we compared the performance of the tuned library routine DAXPY from ESSL on IBM p690 (SAXPY from libsci on CRAY T3E) to the performance of a simple Fortran loop doing the same operation. Here it was interesting to see, whether the optimization of the routine outweighs the overhead of a subroutine call for the small number of operations.
In the other cases we only measured the performance of the library routines as here the number of operations and the complexity is much higher so that a user will in general use the library routine.

3.1 Performance of BLAS 1 DAXPY

The routine DAXPY (on CRAY T3E SAXPY) computes $\vec{y} = \alpha\vec{x} + \vec{y}$ for a scalar α and vectors $\vec{x}$ and $\vec{y}$ of 64-Bit real numbers.
The $nmin$, $nmax$, $ninc$, and $maxtest$ values can be found in table 1 in the appendix.
The execution times on IBM p690 are in the order of 1.0×10^{-7} sec for $n < 100$ and 2.0×10^{-4} for $n = 100000$. This means that $maxtest$ has to be chosen rather large. On CRAY T3E they are a little longer but not a complete order of magnitude, thus $maxtest$ most often was the same for IBM p690 and CRAY T3E.
On IBM p690 $n = 128$ was the largest n where at least the median and the best MFLOP rate for the Fortran code were better than the MFLOP rates achieved with the library routine. For larger n always the routine DAXPY from ESSL was faster than the Fortran code.
This is in contrast to CRAY T3E where the library routine SAXPY from libsci is only faster for $50 \leq n \leq 6208$, for larger n the Fortran code becomes much faster as it always achieves about 300 MFLOPS (see figure 4). The library routine gets really good performance for $640 \leq n \leq 4256$, where it almost always exceeds 500 MFLOPS, which is more than 40 % of the peak performance, for larger n performance decreases from 460 MFLOPS for $4300 \leq n \leq 4480$ to around 430 MFLOPS for $n \leq 6144$, which still is about 36 % of the peak performance. For larger n the performance breaks down and remains almost constant at 62 MFLOPS for $n \geq 8000$, which is now only 5 % of the peak performance.
In contrast to this behavior the Fortran coded AXPY has an almost constant performance of about 300 MFLOPS, which is 25 % of the peak performance. This shows that libsci is optimized for small up to medium size vector lengths. Larger vectors will usually not occur on a single T3E processor as in a typical application also matrices of the same size are used and the small memory of a single T3E processor does not allow very large matrices on a single processor.

On the other side the routine DAXPY from ESSL is optimized for short and long vectors. It reaches its best performance at $n = 1000$ with 1983 MFLOPS, which is less than 30 % of the peak performance of 6800 MFLOPS. For larger n it decreases in three steps. The first one is a decrease to about 1700 MFLOPS ($\approx$ 25 %) for

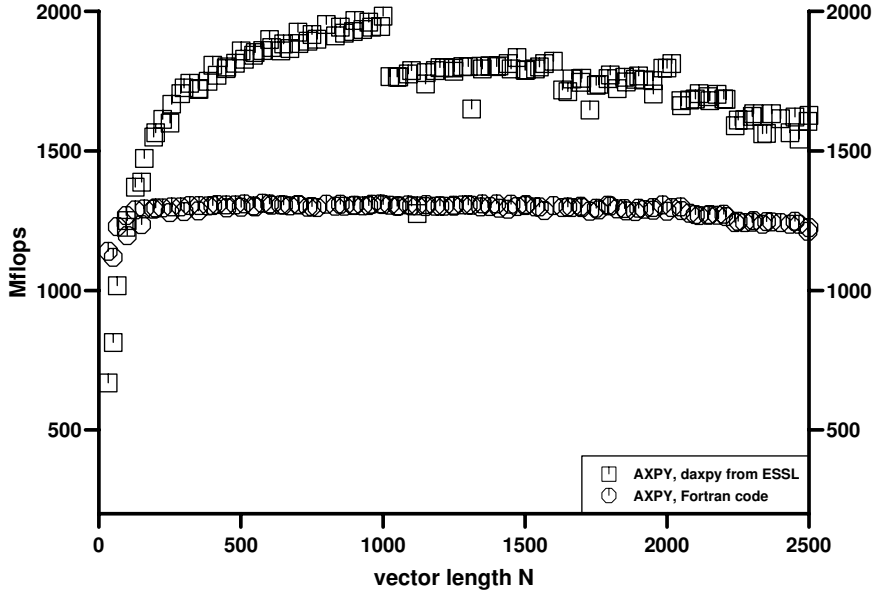

Figure 2: Performance of DAXPY on one processor of IBM p690. For matrix sizes between 32 and 2500 only the highest MFLOP rate achieved is shown. Additionally the same values are shown for a Fortran coded AXPY loop.

$1000 < n \leq 2000$ (see figure 2). This is less than the 25 % which the Fortran loop achieves on CRAY T3E.

Then the performance of the ESSL routine varies between 1400 MFLOPS and 1600 MFLOPS for $2000 \leq n \leq 61000$, which is 20-23 % of its peak performance and then it decreases to less than 1000 MFLOPS (less than 15 % peak) (see figure 3).

The fact that even in the best tuned region less than 30 % of the peak performance of the CPU is achieved shows that the IBM power processor is less balanced concerning CPU speed and memory access speed than the ALPHA processor in CRAY T3E where more than 40 % of the peak performance can be achieved at least for small problems and where the simple Fortran code always achieves about 25 % of the peak CPU performance.

On IBM p690 the Fortran loop achieves only 1300 MFLOPS for $200 \leq n \leq 2000$ (≈ 19 % of the peak performance) and then gradually decreases to 1200 MFLOPS

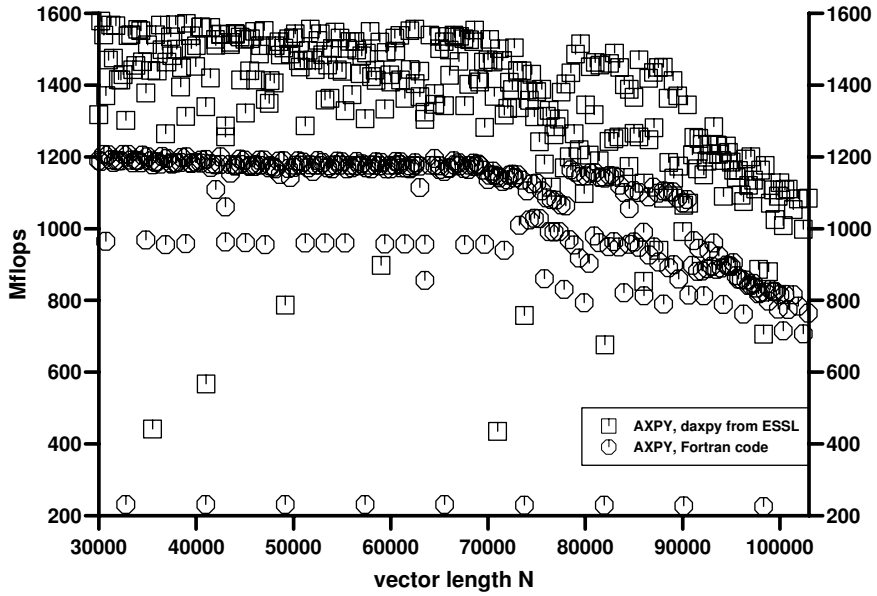

Figure 3: Performance of DAXPY on one processor of IBM p690. For matrix sizes between 30000 and 103000 only the highest MFLOP rate achieved is shown. Additionally the same values are shown for a Fortran coded AXPY loop.

(< 18 % of the peak performance).
For $n > 2000$ the performance of the Fortran loop on IBM remains almost constant at 1200 MFLOPS until $n = 60938$ and then decreases. At $n = 103242$ it is ≈ 800 MFLOPS but it still decreases. We did not measure larger values of n, but W. Frings measured the general triad for a memory test (see [8]) and there it can be seen that it further decreases.
The DAXPY routine from ESSL as well as the Fortran code have several outliers where significantly slower performance is achieved. For the Fortran code the outliers show a more regular pattern than for the ESSL-routine. For all multiples of 8192 the performance of the Fortran loop is around 231 MFLOPS and even less for $n \geq 73728$. For n being a multiple of 24576 also the ESSL-routine showed a performance of less than 800 MFLOPS, for the other multiples of 8192 the ESSL-routine reached between 1300 and 1400 MFLOPS for $n \leq 65536$ and ≈ 1100 MFLOPS for $n \geq 81920$. For values of n close to multiples of 8192 there were

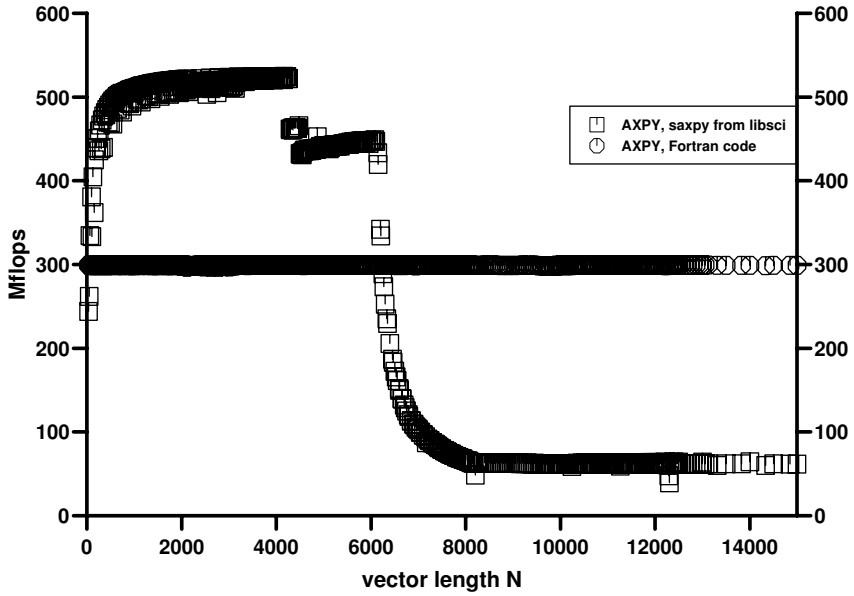

Figure 4: Performance of SAXPY on one processor CRAY T3E. For matrix sizes between 32 and 115000 the highest MFLOP rate is shown. Additionally the MFLOP rates are shown for a Fortran coded AXPY loop.

sometimes larger break-ins, the largest one was for $n = 8200$, where the ESSL-routine DAXPY had its slowest performance measured: The highest MFLOP rate achieved with DAXPY for $n = 8200$ was 300 MFLOPS (the Fortran code reached 972 MFLOPS) and for $n = 8250, 8256,$ and 8300 still MFLOP rates of 796 to 872 were achieved.

The outliers with ESSL where even the best performance was less then 1000 MFLOPS were in different regions and at some isolated points. The slowest performance was achieved for $n = 12300,\ 20500,\ 41000,\ 35500,$ and 71000. For the last two values less than 500 MFLOPS were achieved, in the other cases between 500 and 600 MFLOPS were achieved. For the first two values with the Fortran code about 800 MFLOPS were achieved, for the other values between 1150 and 1200 MFLOPS were achieved with the Fortran code.

3.2 Performance of BLAS 2 DGEMV

The routine DGEMV (SGEMV for CRAY T3E) computes $\vec{y} = \alpha\, op(A)\vec{x} + \beta\vec{y}$, where α and β are scalars, $\vec{x}$ and $\vec{y}$ are n-vectors of 64-Bit reals and $op(A)$ is either A or A^T with an $n \times n$-matrix A of 64-Bit reals.
We measured performance with A not transposed and A transposed. The sizes of n, $ninc$, and $maxtest$ can be seen from table 2 in the appendix.
The largest problem size on CRAY T3E was $n = 8000$ because of the small memory per processor. Due to the larger memory, using 64-Bit addressing mode on IBM p690 we could go up to $n = 12500$.

On IBM p690 the execution times for $n < 100$ were less than 1.e-5 sec, for

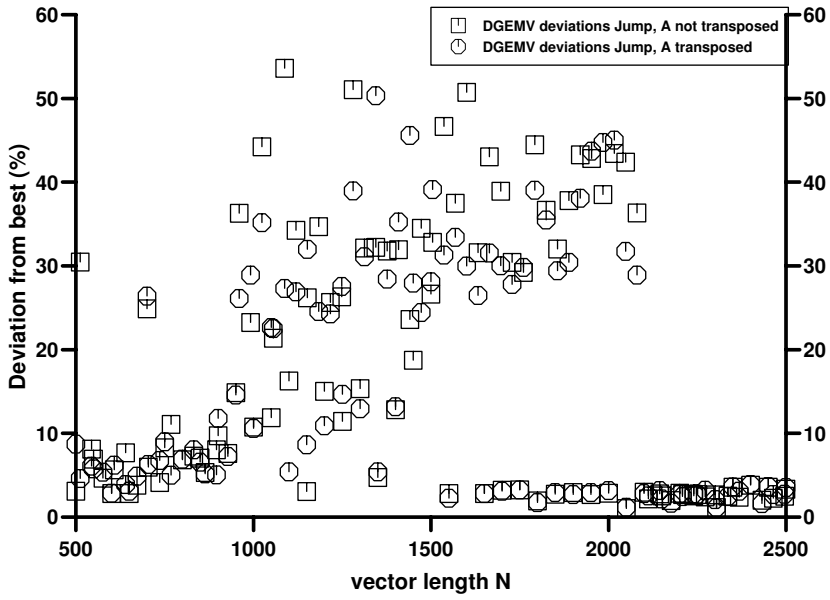

Figure 5: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 500 and 2500 the deviation of the slowest execution time from the fastest execution time is shown for A transposed and not transposed.

$n < 320$ less than 1.e-4 sec, for $n \leq 750$ less than 1.e-3 sec, for $n \leq 2016$ less than 1.e-2 sec and for $2016 \leq n \leq 2656$ still at least the shortest execution times were often less than 1.e-2 sec. For $n > 5700$ all execution times were longer than

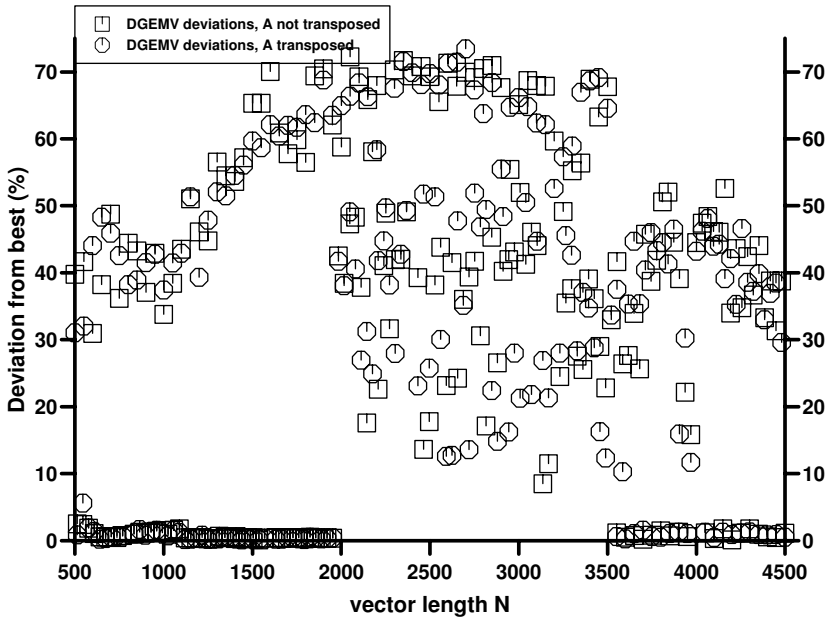

Figure 6: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 500 and 4500 the deviation of the slowest execution time from the fastest execution time is shown for A transposed and not transposed on a dedicated part of Jump.

0.1 sec, but even for $n = 12500$ the execution times on IBM p690 were shorter than 0.5 sec. On CRAY T3E the execution times for $n \geq 7000$ were in the order of 1 sec.

Again there were large differences between several measurements of the same problem size on IBM p690 and only small differences on CRAY T3E. So again we did at least three measurements per problem size on IBM p690 and often only one on CRAY.

The highest deviations were found in the range $960 \leq n \leq 2080$ where for A transposed as well as for A not transposed the deviations of the slowest execution time from the fastest was in most cases higher than 10 % (see figure 5). For $n \geq 2100$ the deviation is most often smaller than 5 % or in some regions between 5 % and 10 % until it again exceeds 10 % for $n \geq 6080$, but this time it exceeds 20 % only in some rare cases.

To find out whether these large deviations are due to other users we also did the

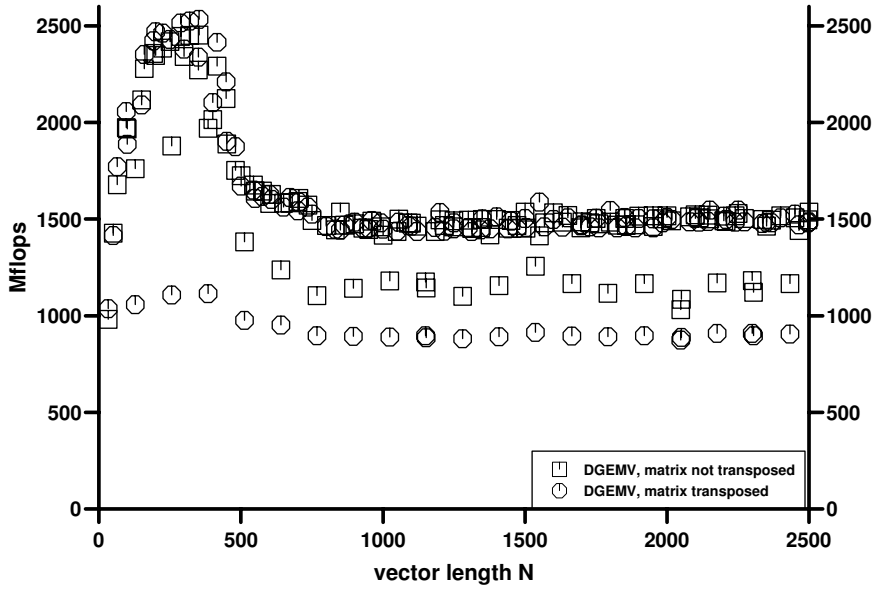

Figure 7: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 32 and 2500 the best MFLOP rate measured is shown for A transposed and not transposed.

measurement in a dedicated environment. Due to a typo in our execution script we did the measurements 24 times in one night for n being a multiple of 50 between 450 and 3500, thus we have 24-48 timing results for each value in that range. With so many measurements it is seen that for $n \geq 650$ (n being a multiple of 50) at least one of the runs achieved less than 1000 MFLOPS with A transposed as well as with A not transposed. (It was not always the same run for A not transposed and A transposed.) For $n \geq 1300$ at least one of 48 runs achieved less than 50 % of the performance of the fastest run, for $n \geq 1550$ there were at least two out of 24 runs with less than 50 % of the best achieved performance. There is just one exception, for $n = 2250$ each run achieved more than 50 % of the performance of the best run.

For n between 2000 and 3500 we even observed very slow performance for n being a multiple of 32, and only one or two runs sometimes achieved the

performance observed on the non-dedicated part of Jump. This lead to the high deviations for all values of n between 2000 and 3500 on the dedicated part of Jump (see figure 6).

This shows that we can not explain the differences in performance by other

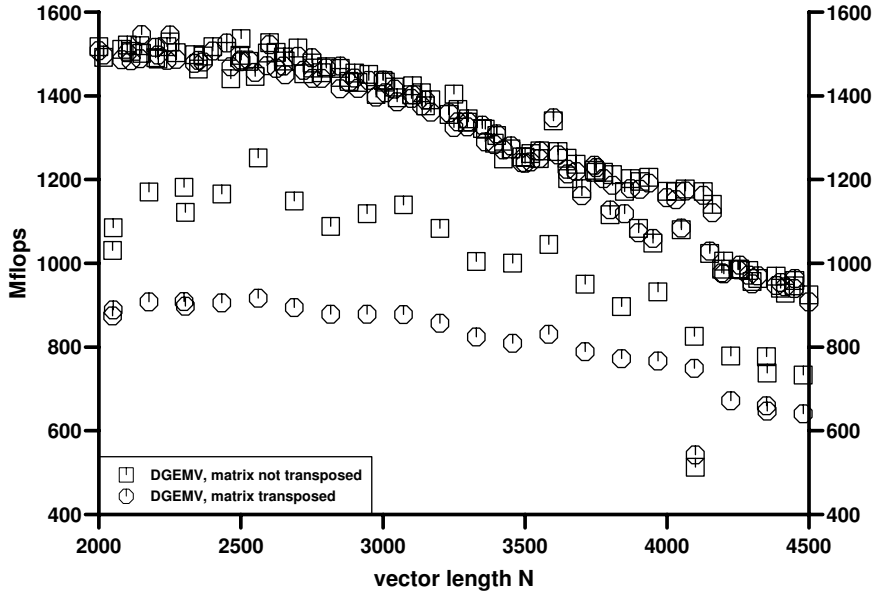

Figure 8: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 2000 and 14500 the best MFLOP rate measured is shown for A transposed and not transposed.

user programs sharing the memory and other resources. On the other hand it must be some memory access effect, because it does not appear with the smallest problems which fit into the cache and thus reach a higher performance than the other problems.

Due to the problems with the non reproducible results we will only show figures with the best MFLOP rates reached, those with best, median and worst are too much filled.

The best performance on IBM p690 is achieved for $150 \leq n \leq 448$ with the exceptions $n = 256$ and $n = 384$. In all other cases more than 2000 MFLOPS (30 % of peak) were reached. For $n = 250, 288, 320$, and 352 even more than

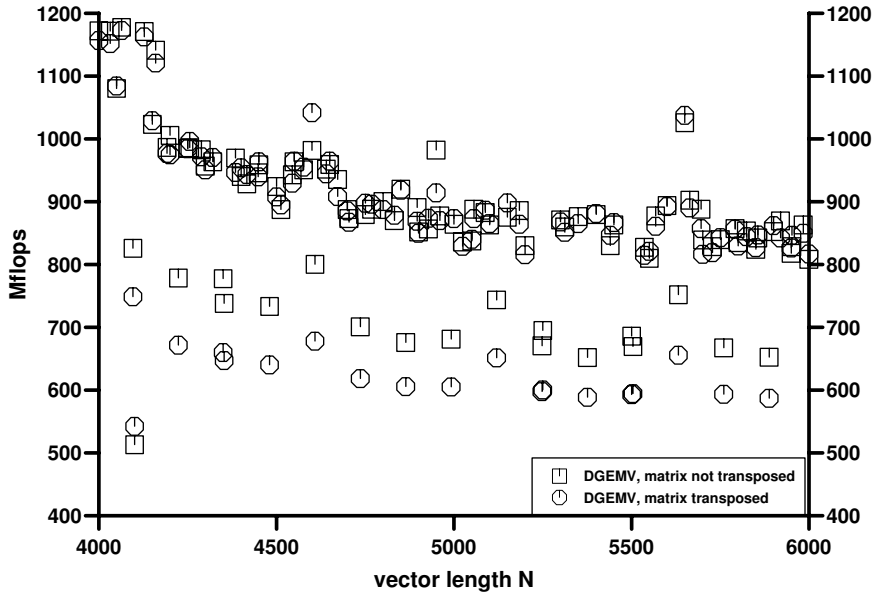

Figure 9: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 4000 and 6000 the best MFLOP rate measured is shown for A transposed and not transposed.

2400 MFLOPS (35 % of peak) were achieved when A was not transposed. With A transposed even more than 2500 MFLOPS (37 % of peak) were achieved for $n = 288, 320$, and 352.On the other side for $n = 256$ with A not transposed 1878 MFLOPS were achieved, with A transposed only 1107 MFLOPS, and for $n = 384$ with A not transposed 1971 MFLOPS and with A transposed only 1115 MFLOPS were achieved (see figure 7).

When the matrix A becomes larger the performance decreases to about 1500 MFLOPS for $800 \leq n \leq 2500$ (22 % of peak) with many outliers (see figures 7 and 8). For many of these outliers less than 1000 MFLOPS were achieved even with the best run, and for values of n close to the outliers often at least the worst run also did not achieve 1000 MFLOPS.

The pattern of the outliers seems to be very regular, they always show up at multiples of 128, and the performance with A transposed is worse than with A not transposed. This shows that on IBM p690 matrix sizes which are multiples of 128

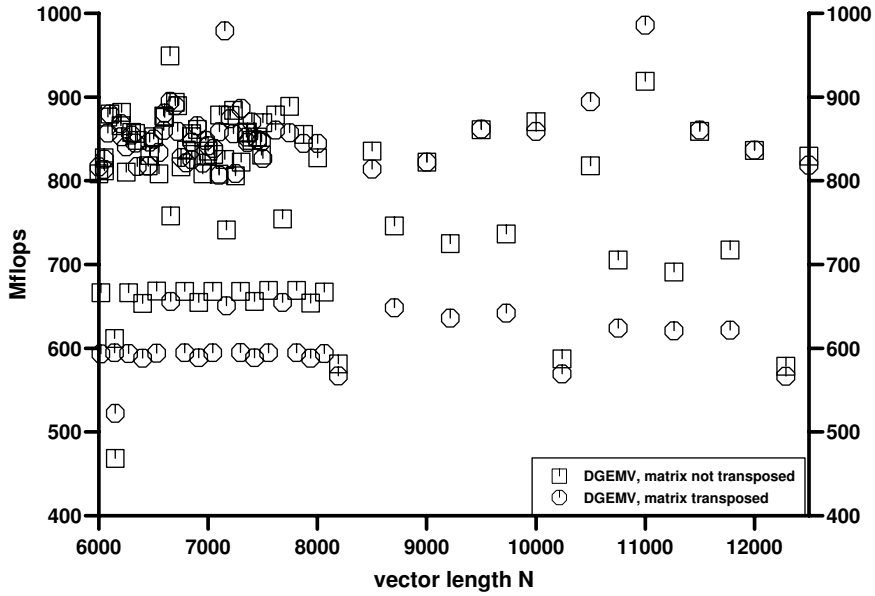

Figure 10: Performance of DGEMV on one processor of IBM p690. For matrix sizes between 6000 and 12500 the best MFLOP rate measured is shown for A transposed and not transposed.

or close to multiples of 128 should be avoided.
For $n > 2500$ the performance of DGEMV on IBM p690 decreases from about 1500 MFLOPS steadily to around 800 MFLOPS to 900 MFLOPS (12 % to 13 % peak) for $n \geq 6000$ being a multiple of 100 and around 600 to 700 % MFLOPS (9 % to 10 % peak) for n being a multiple of 128 (see figures 8 and 9, and 10). This is much less than the performance of DAXPY although it should be better because of the slightly higher data re-use factor.
From figure 11 it can be seen that on CRAY T3E the best performance for SGEMV is achieved for $n = 64$, i.e. for a very small problem, and it is 880 MFLOPS if A is not transposed and 650 MFLOPS if A is transposed. The first rate is more than 70 % of the peak performance and even for A transposed more than 50 % of the peak performance is achieved. But the performance decreases rapidly and reaches an average of 170 MFLOPS slightly decreasing to 140 MFLOPS for large matrices if n is no multiple of 32 and not too close to a multiple of 32. For n being

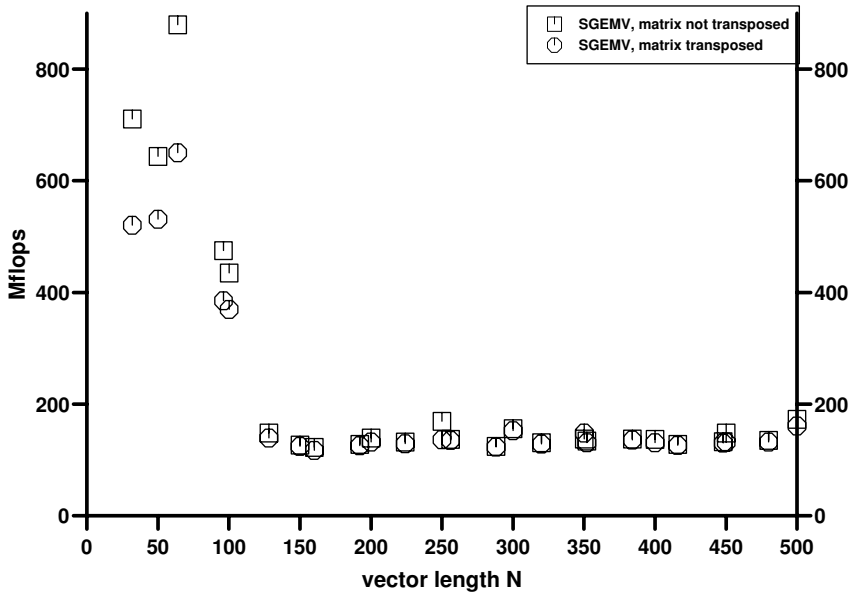

Figure 11: Performance of SGEMV on one processor of CRAY T3E. For matrix sizes between 32 and 500 the best MFLOP rate measured is shown for A transposed and not transposed.

a multiple of 32 only rates from 140 MFLOPS for small matrices decreasing to 120 MFLOPS for large matrices are achieved (see figure 12). This means that for small matrices and optimal problem sizes an average of 15 % of the peak performance is achieved whereas for larger non-optimal matrices an average of 10 % of the peak performance is achieved.

There are several outliers at multiples of 1024, which seems to be a critical problem size to be avoided on CRAY T3E. The smallest rates you get with $n = 4096$ and matrix A transposed: it is only 36 MFLOPS compared to an average of 120 for matrix sizes around 4000. If A is not transposed, the performance is only slightly better, namely 45 MFLOPS. For $n = 4100$ the performance is still rather bad but already more than 60 MFLOPS in both cases. The next minimum is at $n = 2048$ and still at $n = 2050$, this time the case with A not transposed is worst with 56 MFLOPS for $n = 2048$ and 62 MFLOPS for $n = 2050$, whereas the rates for A transposed are 71 and 88 MFLOPS respectively. For $n = 1024$

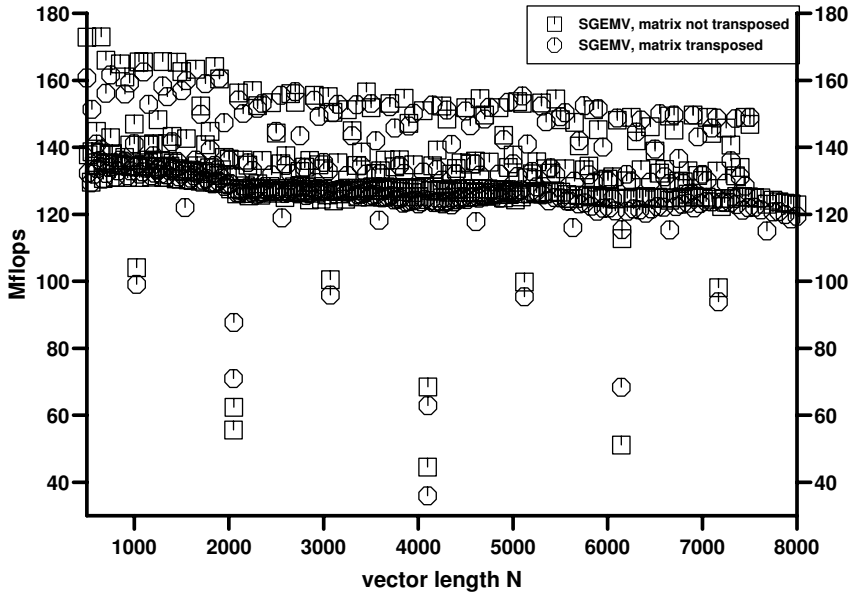

Figure 12: Performance of SGEMV on one processor of CRAY T3E. For matrix sizes between 500 and 8000 the median of the MFLOP rate measured is shown for A transposed and not transposed.

and $n = 3072$ the rates exceed 100 MFLOPS if A is not transposed and exceed 95 MFLOPS for A transposed.
Overall GEMV performs better, i.e. it achieves a higher percentage of the CPU's peak performance on IBM p690 than on CRAY T3E but with more outliers and a much bigger range of execution times for a single problem size. We cannot explain those differences in performance on IBM p690.

3.3 Performance of BLAS 3 DGEMM

The routine DGEMM (SGEMM for CRAY T3E) computes $C = \alpha op(A)op(B) + \beta C$, where $op(A)$ is A or A^T, $op(B)$ is B or B^T, A, B, and C are $n \times n$-matrices and α and β are scalars of 64-Bit reals.
We did measurements for all combinations of A and B not transposed or transposed. The sizes of n, $ninc$, and $maxtest$ can be seen from table 3 in the appendix.
As the matrix multiplication does about n^3 operations, the execution times now grow much faster than for the matrix-vector multiplication and thus on IBM p690 we came to execution times of 1 sec for $n \geq 1300$ and on CRAY T3E for $n \geq 700$. This means that $maxtest$ could be set to 1 for rather small n.

For the BLAS 3 routine DGEMM we did not find the large deviations of the

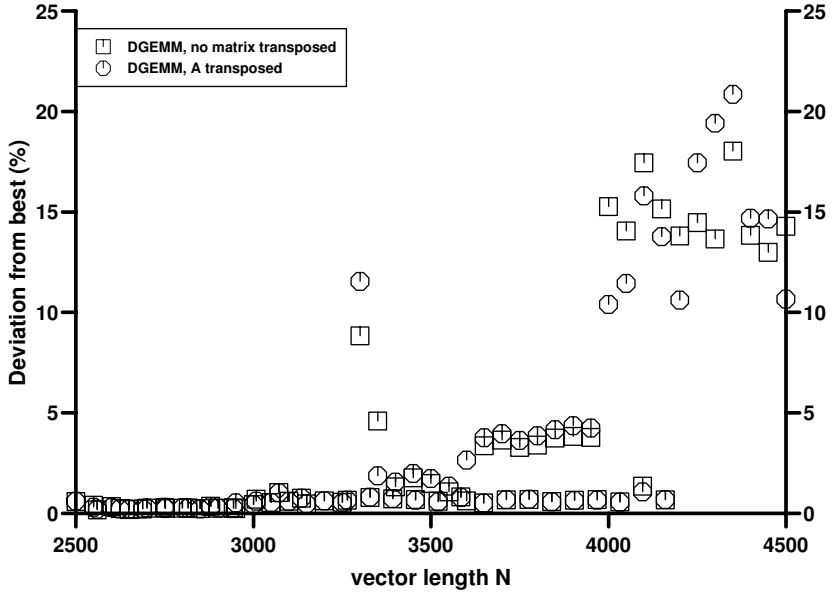

Figure 13: Performance of DGEMM on one processor of IBM p690. For matrix sizes between 2500 and 4500 the deviation of the slowest MFLOP rate from the best MFLOP rate in percent is shown for matrix B not transposed.

slowest execution time from the fastest one. Only for $n = 32$ and only matrix B

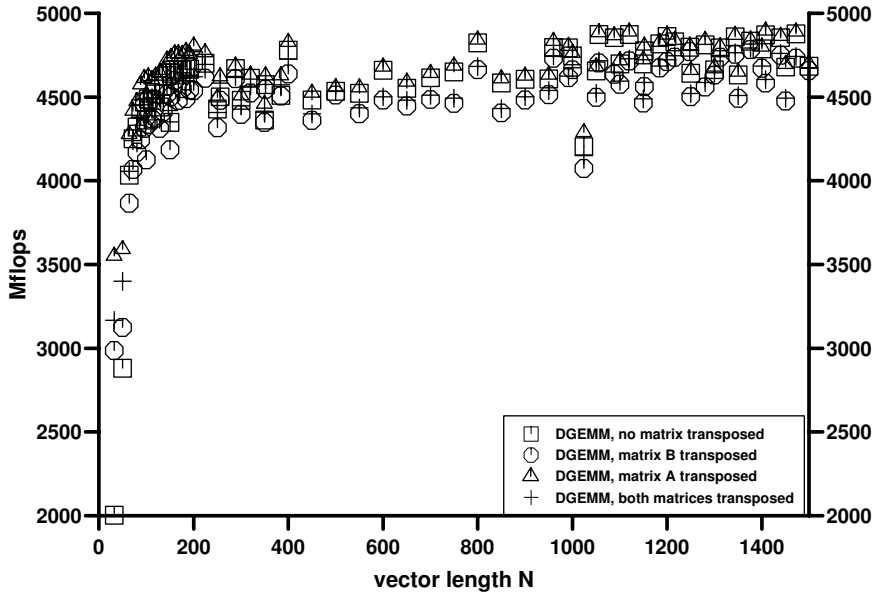

Figure 14: Performance of DGEMM on one processor of IBM p690. For matrix sizes between 32 and 1500 the best MFLOP rate measured is shown for all combinations of transposition.

transposed we had an out-lier where the slowest performance achieved less than 10 % of the best. As this was the only out-lier for $N < 3000$ we do not show this in the figures. For larger n more than 10 % deviation from the shortest execution time could only be seen for problem sizes larger than 3000 (see figure 13). The figures are similar for B transposed and B not transposed thus we only show the case with B not transposed to not overfill the figure.
Again there is a difference for very small problem sizes between IBM and CRAY. On CRAY T3E the best performance is achieved for very small matrices (see figure 17) whereas on IBM p690 the performance increases in the beginning and reaches its peak for a little larger matrices (see figure 14).
On IBM p690 the situation is rather homogeneous, there is no large difference between the cases of transposition. Both cases with B transposed are slightly less performant than the cases where B is not transposed. There are also some performance break-ins if n is a multiple of 1024 or close to it, especially for $n = 2048$,

$n = 2050$ and $n = 4096$, $n = 4100$ (see figures 14, 15, and 16).

In general on IBM p690 the performance of DGEMM increases until $n = 144$,

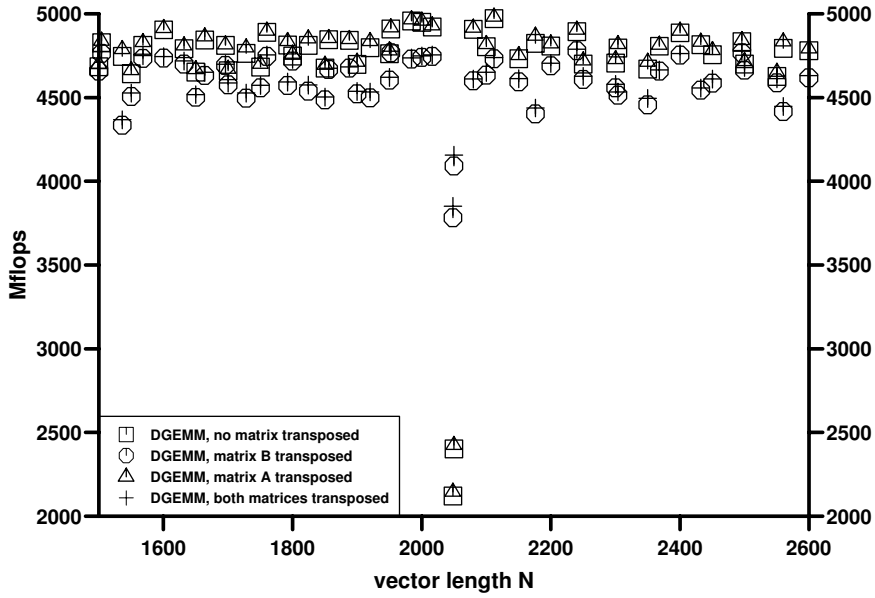

Figure 15: Performance of DGEMM on one processor of IBM p690. For matrix sizes between 1500 and 2600 the best MFLOP rate measured is shown for all combinations of transposition.

where 4600 MFLOPS are exceeded if no matrix is transposed, 4700 MFLOPS if A is transposed, 4600 MFLOPS if both matrices are transposed and 4500 MFLOPS if only B is transposed. From there on the performance lies in the range between 4400 MFLOPS, which is about 65 % of IBM p690's peak performance and less than 5000 MFLOPS, which is about 73 % of the peak performance if B is not transposed and between 4200 MFLOPS, about 62 % of the peak performance, and 4800 MFLOPS, 70 % of the peak performance, if B is transposed.

On CRAY T3E optimal performance is achieved for $n = 64$ where it reaches 1000 MFLOPS if no matrix is transposed, which is 83 % of the peak performance, but the performance decreases rapidly to about 700 MFLOPS in the three cases no matrix transposed, both matrices transposed and only A transposed (see figure 17). In the case where only B is transposed the average performance on CRAY T3E lies

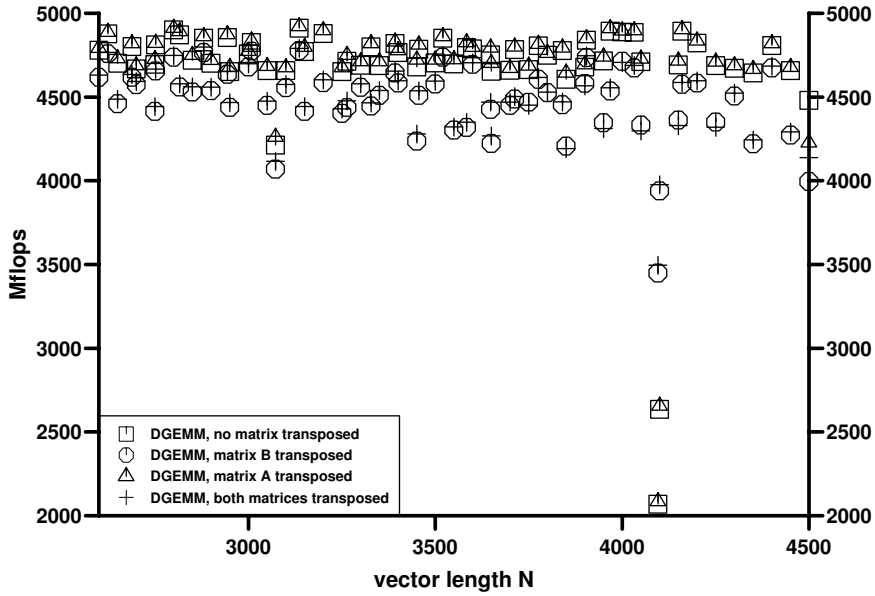

Figure 16: Performance of DGEMM on one processor of IBM p690. For matrix sizes between 2600 and 4500 the best MFLOP rate measured is shown for all combinations of transposition.

at about 500 MFLOPS, and for multiples of 128 it most often achieves less than 100 MFLOPS (see figures 17, 18, and 19).

This means that on CRAY T3E the matrix-matrix-multiplication on average achieves about 58 % of the peak performance except in the case when only matrix B is transposed where it achieves only 41 % of the peak performance in general cases and less than 10 % if n is a multiple of 128.

This shows that the DGEMM routine in ESSL is better tuned than the SGEMM routine in libsci.

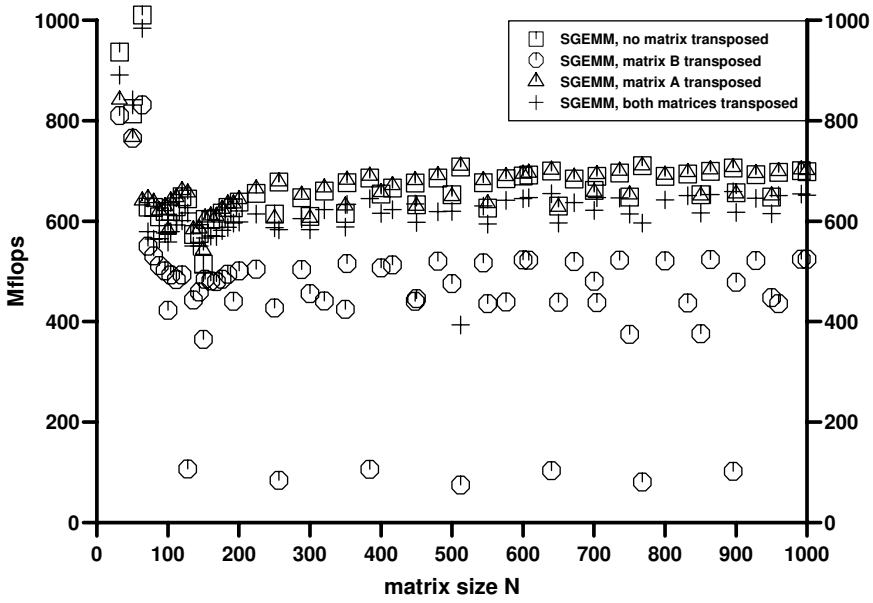

Figure 17: Performance of SGEMM on one processor of CRAY T3E. For matrix sizes between 32 and 1000 the best MFLOP rate measured is shown for all combinations of transposition.

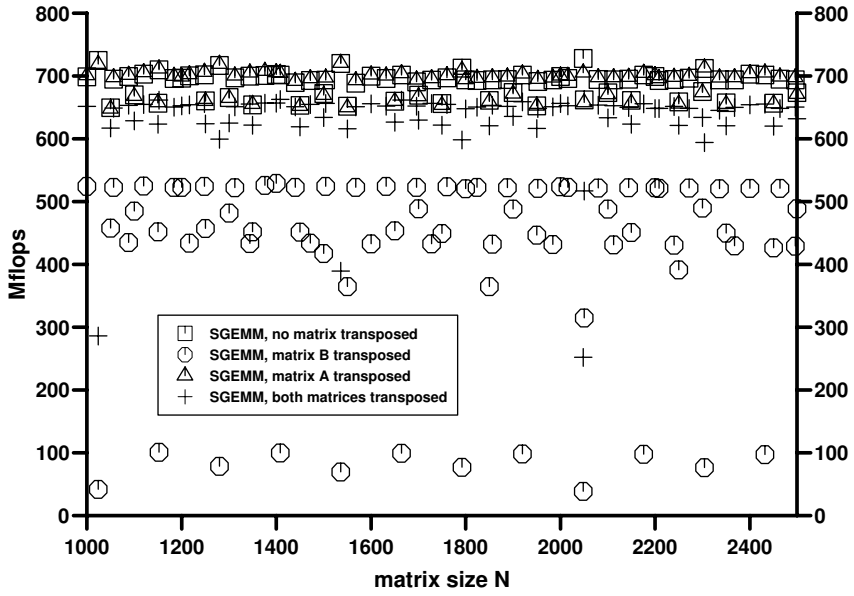

Figure 18: Performance of SGEMM on one processor of CRAY T3E. For matrix sizes between 1000 and 2500 the best MFLOP rate measured is shown for all combinations of transposition.

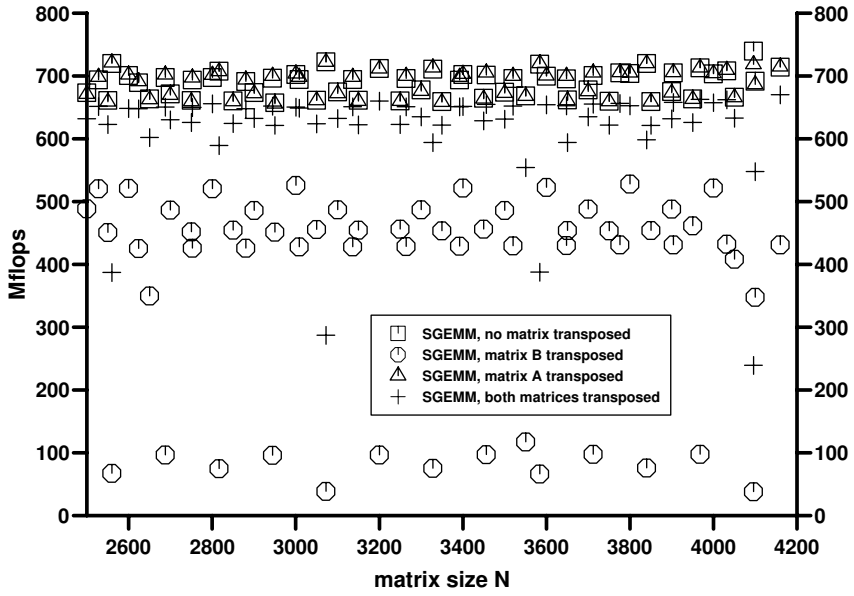

Figure 19: Performance of SGEMM on one processor of CRAY T3E. For matrix sizes between 2500 and 4200 the best MFLOP rate measured is shown for all combinations of transposition.

4 Conclusions

For prediction of the performance of large application programs it is interesting to know the performance of the basic routines used in these programs.
From the results it can be seen that only optimized routines with a data re-use factor in the order of the matrix size can achieve more than 50 % of the peak performance of a cache based computer like IBM p690. Programs based on level 1 BLAS and level 2 BLAS operations can at best achieve about 25 % of the peak performance on IBM p690 and this only for small problems, for larger problems only about 20 % of the peak performance are achieved. To our surprise the BLAS 2 routine DGEMV on an average achieved a smaller part of the peak performance than the BLAS 1 routine DAXPY.
With level 3 BLAS operations about 60 % of the peak performance can be achieved. As the differences in level 1 to level 3 BLAS operations are larger on IBM p690 than it had been on CRAY T3E it is even more important to use blocked algorithms which can make use of level 3 BLAS operations on IBM p690 than it was on CRAY T3E.
Unfortunately there is a rather high uncertainty in performance prediction on IBM p690. Factors which are not due to the program code but perhaps due to the operating system influence the performance. In a non-dedicated environment several users share the memory and thus the memory bandwidth for one user's program is reduced. This can especially reduce the performance of those routines with the smaller cache re-use factor.

References

[1] C.L. Lawson, R.J. Hanson, D.R. Kinkaid, and F.T. Krogh, *Basic linear algebra subprograms for FORTRAN usage*,
ACM Transactions on Mathematical Software 5 (3), pp 308-323, 1979

[2] J.J. Dongarra, J. Du Croz, and R.J. Hanson, *An extended set of FORTRAN basic linear algebra subprograms*,
ACM Transactions on Mathematical Software 14 (1), pp 1-17, 1988

[3] J.J. Dongarra, J. Du Croz, and I. Duff, *A set of level 3 basic linear algebra subprograms*,
ACM Transactions on Mathematical Software 16 (1), pp 1-17, 1990

[4] *ESSL - Engineering and Scientific Subroutine Library for AIX Version 4.1*
http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/essl.html

[5] *ATLAS - Automatically Tuned Linear Algebra Software*
http://www.netlib.org/atlas

[6] B. Lang, *Direct Solvers for Symmetric Eigenvalue Problems*
in *Modern Methods and Algorithms of Quantum Chemistry*, J. Grotendorst (Editor), Proceedings, NIC Series Volume 3, February 2000

[7] *JUelich Multi Processor*
http://jumpdoc.fz-juelich.de

[8] *IBM p690 e-server Cluster 1600 performance measurements*
http://www.fz-juelich.de/zam/files/docs/vortraege/ibm_nutzung/Jump_performance.pdf

Appendix

nmin	nmax	ninc	maxtest Jump	maxtest T3E
32	1088	32	10000000	1000000
50	1500	50	10000000	1000000
960	2080	32	1000000	1000000
1300	2500	50	1000000	1000000
1984	4160	32	100000	10000
2300	5500	50	1000000	100000
4000	5120	32	100000	10000
4032	6080	64	100000	100000
5300	7500	50	100000	10000
6016	8128	64	100000	100000
7300	9500	50	100000	10000
9300	10500	50	100000	10000
9920	12096	64	100000	100000
10300	12500	50	100000	10000
11840	13120	64	100000	100000
10240	21504	512	10000	10000
10500	20500	500	10000	10000
18500	30500	500	10000	10000
19456	31744	512	10000	10000
28500	40500	500	10000	10000
29696	41984	512	10000	10000
38500	50500	500	10000	10000
39936	52224	512	10000	10000
48500	60500	500	10000	1000
50176	62464	512	10000	10000
58500	70500	500	10000	1000
60416	72704	512	10000	10000
68500	80500	500	1000	1000
70656	82944	512	1000	1000
78500	90500	500	1000	1000
80896	93184	512	1000	1000
88500	100500	500	1000	1000
91136	103424	512	1000	1000

Table 1: Vector lengths and number of repetitions for the AXPY measurement on IBM p690 and CRAY T3E.

nmin	nmax	ninc	maxtest Jump	maxtest T3E
32	320	32	100000	10000
32	384	32	100000	10000
50	500	50	10000	10000
64	192	32	100000	10000
352	544	32	10000	1000
450	1500	50	1000	100
480	1088	32	1000	100
960	2080	32	1000	100
1300	2500	50	1000	100
1984	3104	32	100	10
2300	3500	50	100	10
3008	4160	32	100	10
3300	4500	50	100	10
4000	5120	32	100	10
4032	5568	64	10	10
4300	5500	50	100	10
5300	6500	50	100	10
5504	6080	64	10	5
6016	8000	64	10	5
6000	8000	500	10	5
6300	6500	50	10	5
5000	10000	500	5	-
7168	12288	512	5	-
9500	12500	500	5	-

Table 2: Vector lengths and number of repetitions for the GEMV measurement on IBM p690 and CRAY T3E.

nmin	nmax	ninc	maxtest Jump	maxtest T3E
32	384	32	10	10
50	500	50	10	10
64	200	8	100	100
352	1088	32	10	5
450	1500	50	1	1
960	2080	32	1	1
1300	2500	50	1	1
1984	3136	64	1	1
2300	3500	50	1	1
3008	4160	64	1	1
3300	3800	50	1	1
3700	4100	100	1	1

Table 3: Matrix sizes and number of repetitions for the GEMM measurement on IBM p690 and CRAY T3E.